Weighted Voronoi Stippling

Adrian Secord[*]
Department of Computer Science
University of British Columbia, Vancouver, BC, Canada

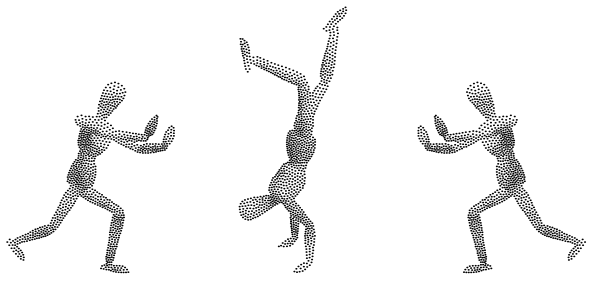


Figure 1: Artist's posable figures with approximately 1000 stipples each

Abstract

The traditional artistic technique of *stippling* places small dots of ink onto paper such that their density give the impression of tone. The artist tightly controls the relative placement of the stipples on the paper to produce even tones and avoid artifacts, leading to long creation times for the drawings.

We present two non-interactive techniques for generating stipple drawings from grayscale images using weighted centroidal Voronoi diagrams. An iterative technique acts on input images directly to produce high-quality stipple drawings and a real-time approach uses precomputed dot distributions to stipple images quickly.

CR Categories: I.3.3 [Picture/Image Generation]: Display algorithms I.3.4 [Graphics Utilities]: Paint systems I.3.5 [Computational Geometry and Object Modelling]: Geometric algorithms, languages and systems

Keywords: Non-photorealistic rendering, stippling, Voronoi diagrams

1 Introduction

Stippling as a technique came into existence to give artists control over the half-toning processes used when printing images in books was a new and difficult task [Jastrzebski 1985]. The technique consists of carefully placing many small dots of ink on paper to approximate different tones. Stipples are placed closer together to form dark regions and further apart to form lighter regions. The stipples must be placed evenly yet randomly so that the human eye does not see spurious patterns that are not a part of the intended impression. The stipples may vary in size and occasionally shape to convey subtle details.

The original advantage of stippling was its ease of reproduction. The half-toning used to print images in books was of highly variable quality and often drawings were drastically resized to meet space requirements. While normal drawings suffered from such treatment stipple drawings retained their attributes more faithfully. In addition, printing a stippled drawing requires only the ability to produce dots of a single colour, making it an inexpensive technique [Wood 1994].

However, stippling has significant artistic merit independent of its utility. The stipples can represent fine detail and texture with little cost in complexity. Stippling is particularly good at clearly representing smooth, rounded objects without sharp edges and so is often used in medical and archaeological texts.

We wish to generate stipple drawings from images with as little user input as possible. The goal is to develop a tool which can generate high-quality stipple drawings from any source whatsoever, which implies that we use images as input and not 3D models. While this limits the amount of information we have to work with, it allows us a greater variety of input sources. For example, a user could start from a scanned pencil sketch, a photograph, the output of a 3D interactive application, frames of an animation, etc.

One of the features of a good stipple drawing is that the stipples are *well-spaced*, that is, the stipples do not clump together, leave uneven voids, or form unwanted patterns. The artist achieves this by carefully placing each stipple onto the page, explaining why stipple drawings often take weeks to create by hand.

Central to our approach is the use of centroidal Voronoi diagrams to produce good distributions of points, as explained in Section 2.1. These distributions can be pre-computed for various different constant tonal values and accessed at run-time to generate stipple drawings rapidly, as covered in Section 4. Alternatively, the input image can be used directly as a weighting function to create a distribution of points that approximate its tones. This method produces images of higher quality but takes more processing time, as explained in Section 3.

[*]ajsecord@cs.ubc.ca, http://www.cs.ubc.ca/~ajsecord

1.1 Related Work

Our iterative method is a direct descendant of the one described in Deussen et al. [2000]. Their method generates stipple drawings by first placing stipples roughly using a dithering algorithm on the input image and then relaxing them using Lloyd's algorithm until they are well-spaced. Lloyd's algorithm was first introduced to computer graphics by McCool and Fiume in [1992] for the generation of sampling point sets. Lloyd's algorithm and the resulting centroidal Voronoi diagrams are explained in section 2.2. However, Deussen et al.'s relaxation step does not take into account the underlying image, resulting in a blurring of image boundaries. The blurring occurs because the relaxation process attempts to space out closely-packed stipples and compress widely-spaced stipples. Since their tool was designed for interactive use with an artist, they solved this problem by having the user confine sets of stipples to fixed regions, which are aligned to important image boundaries. Since the stipples are not allowed to cross the region boundaries during relaxation, the most important edges are preserved. We wish to find an algorithm that will maintain image boundaries without human interaction.

Hausner [2001] uses an approach similar to our iterative technique outlined in Section 3 for aligning the rectangular tiles of a decorative mosaic. Their approach differs significantly from ours in that they must align the tiles' orientation in addition to their position. However, edges that are to be preserved by the algorithm must be entered separately by the user. This makes the algorithm less suitable for a non-interactive application.

2 Voronoi Diagrams

An ordinary Voronoi diagram is formed by a set of points in the plane called the *generators* or *generating points*. Every point in the plane is identified with the generator which is closest to it by some metric. The common choice is to use the Euclidean L_2 distance metric

$$|\mathbf{x}_1 - \mathbf{x}_2| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$ are any two points in the plane. The set of points in the plane identified with a particular generator form that generator's Voronoi region, and the set of Voronoi regions covers the entire plane. Figure 2(a) illustrates a set of generating points and their associated Voronoi regions.

We implemented the fast 3D graphics hardware-based algorithm in Hoff [1999] and originally in [Woo et al. 1997] to compute our Voronoi diagrams. The algorithm draws a set of right cones with their apexes at each generator. The cones all have the same height and are viewed from above the apexes with an orthogonal projection. In addition, each cone is given a unique colour which acts as the generator's identity. Since the cones must intersect if there is more than one generator, the z-buffer determines for each pixel which cone is closer to the viewer and assigns that pixel the appropriate colour value. We can then scan the resulting image and determine which generator is closest to each pixel by using the unique colours. This technique allows us to compute discrete Voronoi diagrams extremely quickly and perform computations on the resulting regions.

2.1 Centroidal Voronoi Diagrams

A *centroidal* Voronoi diagram has the interesting property that each generating point lies exactly on the centroid of its Voronoi region. The centroid of a region is defined as

$$\mathbf{C}_i = \frac{\int_A \mathbf{x}\rho(\mathbf{x})dA}{\int_A \rho(\mathbf{x})dA} \tag{1}$$

where A is the region, $\mathbf{x}$ is the position and $\rho(\mathbf{x})$ is the density function. For a region of constant density ρ, the centroid can be considered as the centre of mass. Figure 2(a) has the centroids of each region marked with small circles.

A centroidal Voronoi diagram is a minimum-energy configuration in the sense that it minimizes $\int_A \rho(\mathbf{x})|\mathbf{C}_i - \mathbf{x}|^2$ [Du et al. 1999]. Practically speaking, a centroidal distribution of points is useful because the points are *well-spaced* in a definite sense. Figure 2(b) shows a centroidal Voronoi diagram.

2.2 Generating Centroidal Voronoi Diagrams

Lloyd's method [Okabe et al. 1992] is an iterative algorithm to generate a centroidal Voronoi diagram from any set of generating points. The algorithm can simply be stated:

Algorithm 1 Lloyd's method

```
while generating points $\mathbf{x}_i$ not converged to centroids do
    Compute the Voronoi diagram of $\mathbf{x}_i$
    Compute the centroids $\mathbf{C}_i$ using equation (1)
    Move each generating point $\mathbf{x}_i$ to its centroid $\mathbf{C}_i$
end while
```

Figure 2(a) relaxes under Lloyd's algorithm to become Figure 2(b). The convergence of Lloyd's algorithm to a centroidal Voronoi diagram has been proven for the one-dimensional case. The higher dimensional cases seem to act similarly in practice, though no proof is known [Du et al. 1999]. There are several different convergence criteria which should be equivalent in the limiting case as the algorithm runs forever. The obvious criterion to use would be that the computed centroids are numerically equal to the generating points. However, for most applications this criterion is far too stringent and it would be perhaps better to look at the average distance moved by all generating points. Since we are interested in generating well-spaced sets of points, we look at the average change in inter-point distance, or equivalently, the average change in Voronoi region area.

2.2.1 Efficient Computation of Centroids

Calculating the centroids requires efficiently evaluating the integrals in equation (1). Since the integrals are over arbitrary Voronoi regions, we convert to iterated integrals and integrate the region row by row. In this manner we can precompute much of the integral.

The denominator of the centroid is transformed as follows:

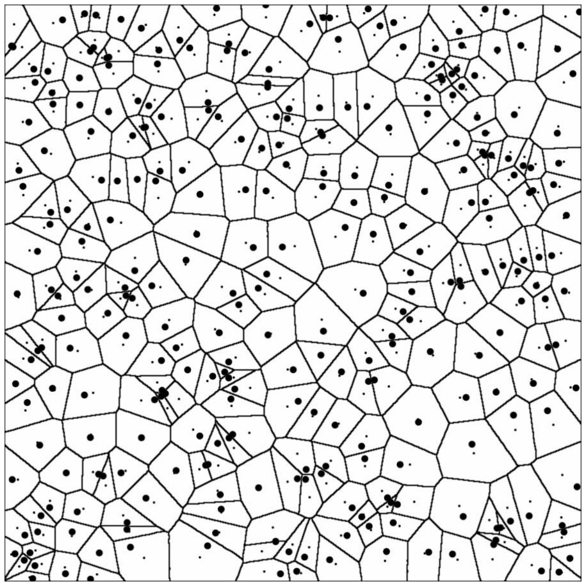
$$\begin{aligned}\int_A \rho(\mathbf{x})dA &= \int_{y_1}^{y_2}\int_{x_1(y)}^{x_2(y)} \rho(x,y)dxdy \\ &= \int_{y_1}^{y_2} [P]_{x_1}^{x_2} dy\end{aligned}$$

where $P \equiv P(x,y) \equiv \int_0^x \rho(s,y)ds$ can be precomputed from the density function[1]. Note that we cannot precompute the entire integral because we do not know the boundaries of the Voronoi regions beforehand.

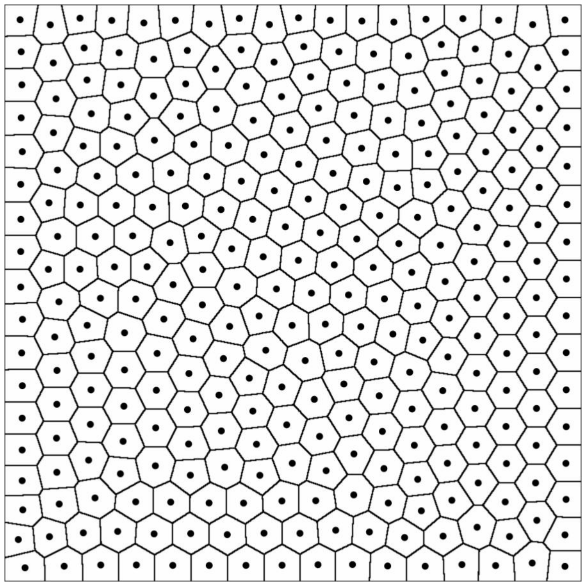
The numerator of the y-coordinate of the centroid is transformed similarly:

$$\begin{aligned}\int_A y\rho(x,y)dA &= \int_{y_1}^{y_2}\int_{x_1(y)}^{x_2(y)} y\rho(x,y)dxdy \\ &= \int_{y_1}^{y_2} y[P]_{x_1}^{x_2} dy\end{aligned}$$

[1]Recall that $[\int_0^x f(s)ds]_a^b = \int_0^b f(s)ds - \int_0^a f(s)ds = \int_a^b f(s)ds$



(a) Voronoi diagram generated by the set of generators (large dots). Centroids of each Voronoi region are marked by the small dots.



(b) Centroidal Voronoi diagram

Figure 2: General and centroidal Voronoi diagrams.

The numerator of the x-coordinate of the centroid involves integration by parts:

$$\begin{aligned}\int_A x\rho(x,y)dA &= \int_{y_1}^{y_2}\int_{x_1(y)}^{x_2(y)} x\rho(x,y)dxdy\\ &= \int_{y_1}^{y_2}\left\{[xP]_{x_1}^{x_2} - \int_{x_1}^{x_2} Pdx\right\}dy\\ &= \int_{y_1}^{y_2}[xP-Q]_{x_1}^{x_2}\,dy\end{aligned}$$

where $Q \equiv Q(x,y) \equiv \int_0^x P(s,y)ds$ can also be precomputed from the density function.

Note that the final expressions require numerical integration only in the y-direction and otherwise involve expressions only at the region boundaries x_1 and x_2. P and Q are precomputed once from the density function and then evaluated at the horizontal end points x_1 and x_2 as needed. This allows us to compute the integrands only at region boundaries and not at every pixel. Otherwise we would have to compute the integrands $x\rho$ and $y\rho$ for every span of pixels across a region and numerically integrated. The above integrand computation is particularly simple – at worst two look-ups for P and Q, a multiplication and a subtraction. In addition, if the Voronoi region is non-convex for numerical reasons, the scan conversion effectively decomposes the region into convex sub-regions, that is, single spans of pixels.

2.3 Resolution of Voronoi Calculation

One disadvantage of using a discrete calculation of the Voronoi regions is the calculation of the centroids is affected by the resolution of the diagram. The relative error of the calculated centroid location will increase as the number of pixels per Voronoi region decreases. A related problem is that if the resolution is low enough, two generating points can effectively overlap and one of the regions will disappear. The solution, as described in [Hoff III et al. 1999], is to split the diagram into tiles and compute each tile at the full resolution available and then stitch the full diagram back together at a higher virtual resolution. The virtual resolution can be increased arbitrarily to meet a lower bound on Voronoi region pixel area.

3 Stippling with Weighted CVDs

The centroidal Voronoi diagrams in Section 2.1 incorporate the idea of a density function $\rho(x,y)$ which weights the centroid calculation. Regions with higher values of ρ will pack generating points closer than regions with lower values. During the iteration of Algorithm 1, the darker regions of the image appear to "attract" more points. We can use Algorithm 1 directly to generate high-quality stippling images by treating a grayscale image as a discrete two-dimensional function $f(x,y)$ where $x,y \in [0,1]$ and $0 \leq f(x,y) \leq 1$ is the range from a black pixel to a white pixel. Define a density function $\rho(x,y) = 1 - f(x,y)$. We can then stipple a given image by first distributing n points in the image and using algorithm (1). Although any distribution of initial points will eventually converge, it is useful to start with a distribution that approximates the final form. Deussen et al. [2000] use a dithering algorithm and we use simple rejection sampling to generate an initial distribution.

3.1 Results

We expect that at the limit of large numbers of very small stipples, the stipple drawing will approximate the grayscale image. Centroidal Voronoi diagrams produce distributions of points that approximate a blue noise distribution, that is, a random distribution with a constraint on the minimum distance between points. Blue noise distributions are useful because they do not introduce spurious patterns such as lines or grids. They can also approximate a constant tone because of the minimum distance constraint. Blue

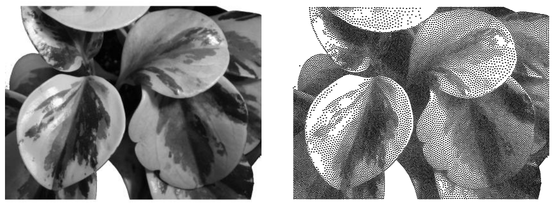


Figure 3: Close-up of large Peperomia leaves with 20000 stipples of radius 2×10^{-3}

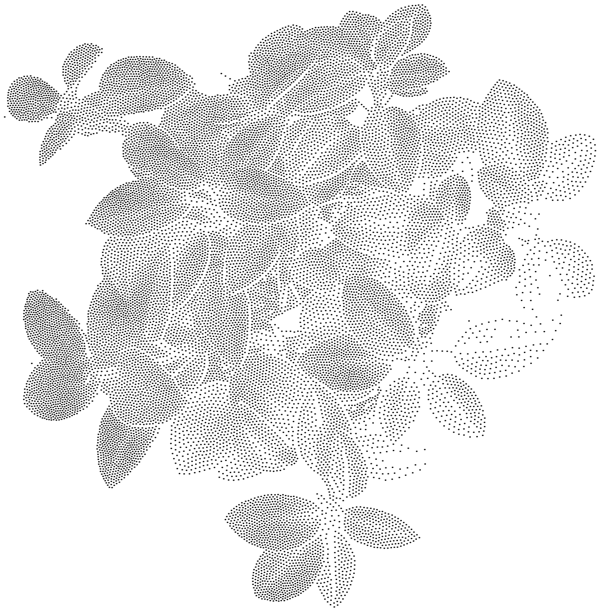


Figure 4: Small Peperomia plant, lit brightly from the right, with 20000 stipples of radius 1.0×10^{-3}

noise distributions have been used to create very high-quality dither patterns for colour reduction [Ulichney 1988]. Figure 3 shows a grey-scale close-up image of some Peperomia leaves with a drawing of 20000 stipples. The fine stippling approximates the tones of the image very well, including the textures inside the leaves.

Figure 4 shows a different small Peperomia plant, lit from the side, with 20000 stipples. Although the number of stipples per square inch is less than in Figure 3, the large number of stipples still renders a faithful image. In particular, note the hard edges maintained by the stipple drawing. Figure 5 shows the full Peperomia plant from Figure 3 with 20000 stipples. Observe the colouration of the centre of the leaf facing the viewer. While the method of Deussen et al. can easily produce sharp edges through user interaction, producing the gradual change in tone visible on the leaf would be difficult. The even spacing of points along the edges of the leaves is the result of the interaction of the centroidal Voronoi diagram, which attempts to space all points evenly, and the density function ρ, which restricts points to the essentially one-dimensional edge.

However, a more interesting test is to apply the method with low stipple counts. Smaller numbers of stipples mean that we cannot rely upon the eye to fuse the tiny size and spacing of the dots into a continuous tone. Figure 6 shows an image of an artist's mannequin

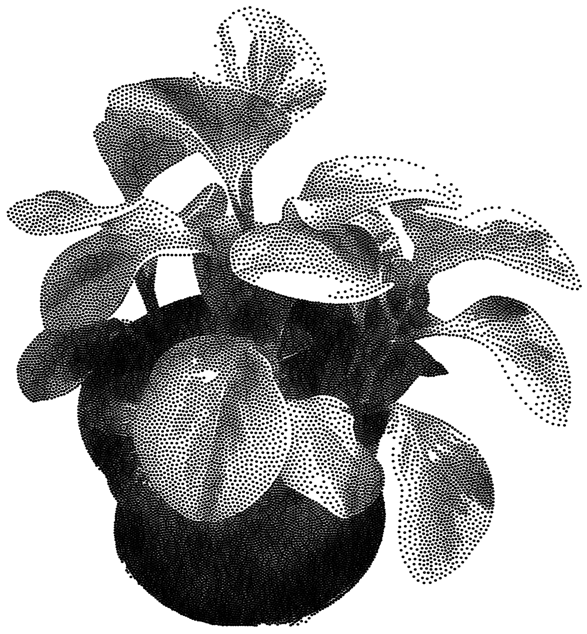


Figure 5: Large Peperomia plant with 20000 stipples of radius 2×10^{-3}

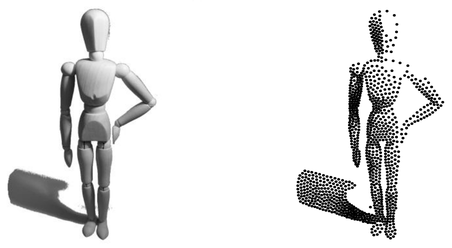


Figure 6: Figure with 1000 stipples of radius 5×10^{-3}

and the stippled version with 1000 stipples. Figure 7 shows a climbing shoe in the same format. Note that both the stipple drawings are quite recognizable, especially in comparison to Figure 8, where the source images have been reduced in resolution until they contain approximately 1000 pixels each[2].

[2]This comparison is not quite fair, as the 1000 pixels are forced to be equally spread across the image whereas the stipples are free to move. The point is that the stippling maintains edges and silhouettes even at very low resolutions.

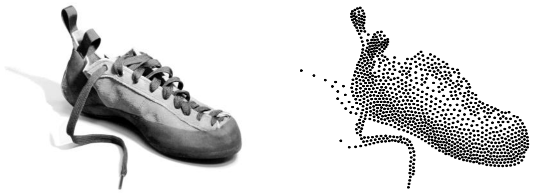


Figure 7: Climbing shoe with 1000 stipples of radius 5×10^{-3}

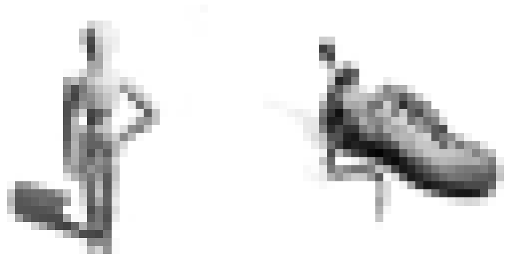


Figure 8: Source images of Figures 6 and 7 rendered with approximately 1000 pixels instead of stipples

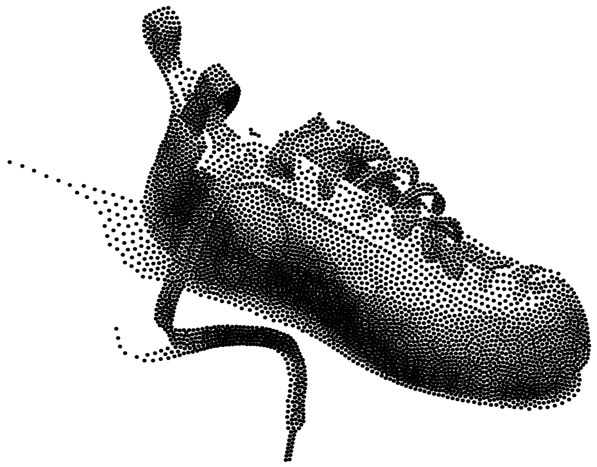


Figure 9: Climbing shoe with 5000 stipples of radius 3×10^{-3}

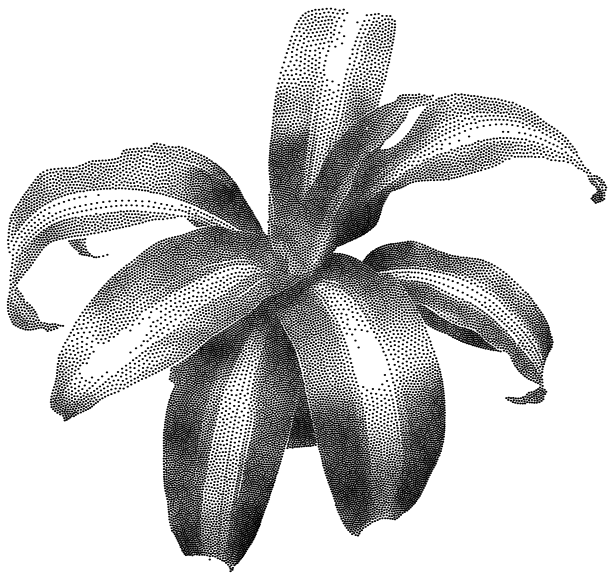


Figure 10: Corn plant with 20000 stipples of radius 1.5×10^{-3}

Finally, we note that the most striking drawings come from neither very-high nor very-low numbers of stipples, but medium ranges. Figure 9 shows the climbing shoe of Figure 7 rendered with 5000 stipples. This drawing seems to both reproduce the range of tones from the original and have the "feel" of a real stipple drawing. Figure 10 shows a corn plant rendered with 20000 stipples and displaying both colouration on the leaves and sharp boundaries on the edges. We feel that this image begins to live up to the quote by Hodges in [1989], page 111, in which he attests to the vibrancy of stippled images: "Like a pointallist painting, the drawing will appear to vibrate slightly."

3.2 Parameters and Timings

We computed all the stipple drawings of Section 3 on an Intel Pentium III 1000 MHz machine with 256 Mb of RAM and a NVIDIA GeForce2 MX graphics accelerator. As discussed in Section 2.3, we require the Voronoi regions to have an average area of at least 500 pixels, which forces a virtual resolution of up to 3600 by 3600 pixels for the 20000 stipple drawings. Since we precompute the integrals P and Q from Section 2.2.1 at full virtual resolution, this requires upwards of 100 Mb of memory. The memory requirement could be reduced by an order of magnitude by computing the integrals in tiles in the same way that the Voronoi diagrams are computed, but this did not seem necessary.

The iterations were stopped and the stipple drawing output when the difference in the standard deviation of the area of the Voronoi regions was less than 1×10^{-4}. Because the background of the input images was not always pure white, stipples were only output if the input image value at that location was greater than 99% of pure white.

On the system used, the stipple drawings with up to 5000 stipples completed in under a minute and the drawings with 40000 stipples complete in about 20 minutes on an otherwise unloaded machine. The 1×10^{-4} stopping limit was arbitrarily chosen and different values will lead to different runtimes.

4 Precomputing Stipple Levels

The method presented in Section 3 can produce excellent stipple drawings given enough time for algorithm (1) to converge. Clearly a faster algorithm is needed to compute stipple drawings at interactive rates. We can accomplish this, albeit at a cost in quality, by precomputing sets of stipples and stitching them together at runtime.

4.1 Stipple Levels

We will call the result of stippling an image of constant tone t the *t stipple level*, $0 \leq t \leq 1$. To generate the t stipple level, we simply use the method of Section 3 with $\rho = 1$ and $\frac{N}{1-t}$ stipples, where N is the number of stipples required in a pure black image[3,4]. Figure 11 shows nine stipple levels from black to white of a distribution of 1000 stipples, each differing by 125 stipples. Typically we would

[3]We can actually set ρ to any constant value at all, since equation (1) is insensitive to scalings of ρ.

[4]The number of stipples required for a pure black image can be computed by considering the optimal hexagonal packing of discs in the plane and expanding their radii until they completely overlap.

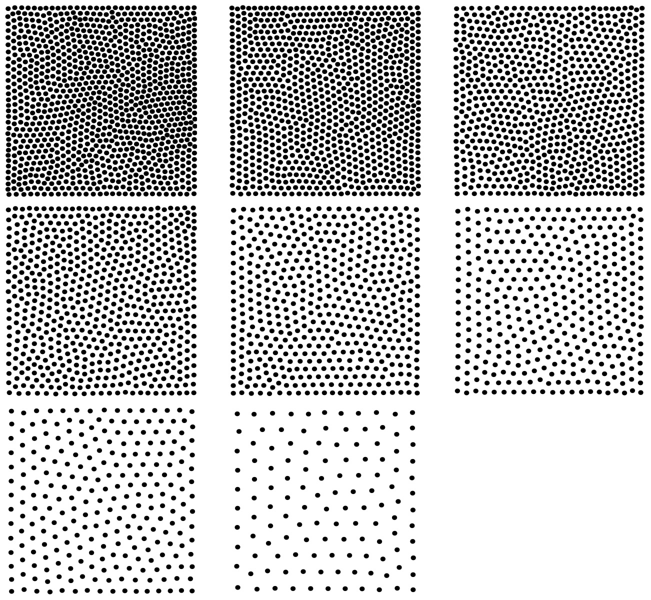


Figure 11: Nine discrete stipple levels with a maximum stipple count of 1000, differing by 125 stipples. The radii of the centre stipple level has been calibrated to represent a 50% gray.

generate a greater number of levels, say 256. We discretise the normally continuous range of tones achievable into a limited number of fixed stipple distributions.

4.2 Fast Stipplings

Using the precomputed stipple levels, we can quickly stipple an image using the following algorithm:

Algorithm 2 Discrete Stippling

```
for all pixel positions $(x,y) \in [0,1] \times [0,1]$ do
    Map image value at $(x,y)$ to stipple level $l$
    Copy stipples on level $l$ inside $(x-\frac{1}{2}, y-\frac{1}{2}) \times (x+\frac{1}{2}, y+\frac{1}{2})$
    to output
end for
```

Algorithm 2 examines the value of each pixel, determines which stipple level is appropriate, and copies all the stipples that fall inside the area covered by the pixel to the output. Since the input image is processed in scan-line order, we sort the stipples in a particular level into bins that cover a single row of pixels, and then sort the stipples in each bin from low x values to high. Given a particular pixel and a particular level this allows us to quickly find the appropriate stipples.

Conceptually, we split the image into a number of regions to be represented by a single stipple level, then stipple each region individually and recompose the stippled regions into a final drawing. The algorithm is quite fast, but is limited by the amount of memory which must be scanned to produce a stipple drawing. Table 1 shows approximate timings on the system described in Section 3.2 for a simple animation loop. The animation in this case was rendered by OpenGL and read back from its buffers, illustrating the flexibility of using images as input. While increasing the numbers of stipples rendered does have a negative effect on the speed, the greatest factor is the image resolution.

Figure 12 compares the results of the fast algorithm to the high-quality algorithm. On the left are the fast stipple drawings of a black-to-white ramp and a lit sphere and on the right are the high-quality versions. Note on the left the many voids and overlapping stipples on the left that introduce spurious detail. They are the result of two regions of the image being stippled with two different stipple levels. The stipple levels cannot merge smoothly since they have different densities of stipples. The result is a pattern that is not as smooth as it should be.

	5000	10000	20000	40000
100×100	350 fps	300	200	150
300×300	150 fps	120	100	80
600×600	60 fps	45	40	35
900×900	20 fps	20	20	18

Table 1: Frames per second at various numbers of stipples and resolutions

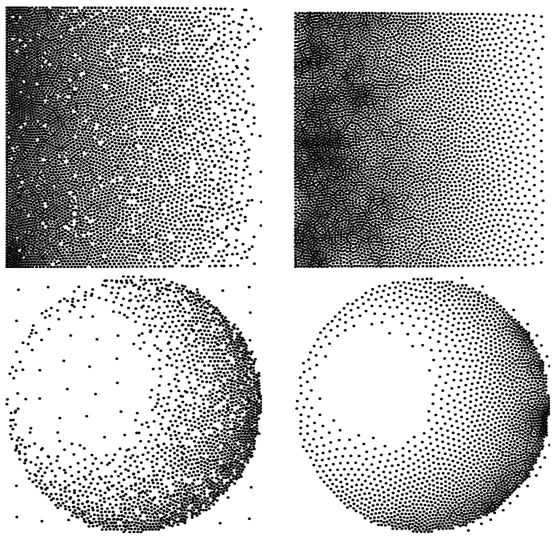


Figure 12: A black-to-white ramp and a lit sphere stippled with the fast algorithm of Section 4.2 on the left and the high-quality algorithm of Section 3 on the right.

In addition, what can not be seen from Figure 12 is the temporal discontinuities that arise when the method is used to stipple an animation. In areas of the image where the tonal value is changing quickly, the pixels get stippled by many different stipple levels in a short time. Even if great care is taken to minimize the differences between one stipple level and the next, the rate at which the pixels change cause them to "shimmer." These problems are minimized by using greater numbers of smaller stipples in the animation.

5 Conclusions and Future Work

We have extended the work on stippling algorithms by introducing a scheme based on weighted centroidal Voronoi diagrams. The presented algorithm has very few user-specified parameters and requires no user interaction. In addition, the input data are grayscale images which can be produced by a wide variety of sources. Apart from simply requiring less work to generate a given stipple drawing, this independence allows cheap stippling to be used in a wider variety of situations than before.

The extension to precomputed stipple levels and thus real-time performance exacted a heavy cost in terms of visual quality, both in still images and in terms of inter-frame coherence of animations. The stipple levels are similar in concept to Praun et al.'s Tonal Art Maps (TAMs) used to hatch 3D objects in [2001]. It should be straight-forward to generate high-quality stippling TAMs and use their approach to investigate frame-coherent animation. However, this would require abandoning our general image-based approach for 3D models.

The tone generated by a set of stipples is problematic and should be investigated further. All the results in this paper use a rational yet ad-hoc method to set the constant radii of the stipples in a particular image. A better understanding of the relationship between stipple radius, spacing and perhaps colour and the resulting perceived tone is required.

In addition, several interesting extensions to the current algorithm could be investigated, including varying the size of the stipples in a single drawing, and the use of colour stipples. Partially transparent blended stipples could be used to alleviate some of the problems with stippling animations.

6 Acknowledgements

The author would like to acknowledge the extremely helpful nature of the reviewers' comments.

References

DEUSSEN, O., HILLER, S., VAN OVERVELD, C., AND STROTHOTTE, T. 2000. Floating Points: A Method for Computing Stipple Drawings. *Computer Graphics Forum 19*, 3 (August).

DU, Q., FABER, V., AND GUNZBURGER, M. 1999. Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM Review 41*, 4 (Dec.), 637–676.

HAUSNER, A. 2001. Simulating Decorative Mosaics. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 573–578.

HODGES, E., Ed. 1989. *The Guild Handbook of Scientific Illustration*. Van Nostrand Reinhold.

HOFF III, K., CULVER, T., KEYSER, J., LIN, M., AND MANOCHA, D. 1999. Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware. In *Proceedings of SIGGRAPH 99*, ACM Press / ACM SIGGRAPH, New York, A. Rockwood, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 277–286.

JASTRZEBSKI, Z. 1985. *Scientific Illustration*. Prentice-Hall.

MCCOOL, M., AND FIUME, E. 1992. Hierarchical poisson disk sampling distributions. In *Proc. of the Graphics Interface '92*, 94–105.

OKABE, A., BOOTS, B., AND SUGIHARA, K. 1992. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley & Sons.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-Time Hatching. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 579–584.

ULICHNEY, R. A. 1988. Dithering with blue noise. *Proceedings of the IEEE 76*, 1, 56–79.

WOO, M., NEIDER, J., AND DAVIS, T. 1997. *OpenGL®Programming Guide*, second ed. Addison-Wesley.

WOOD, P. 1994. *Scientific Illustration*, second ed. Van Nostrand Reinhold.